User's Manual

CC-RL

C++

User's Manual

Applicable Revision
V1.14.00

Renesas Electronics
www.renesas.com

Rev.1.02   Apr 2024

Table of Contents

1. OUTLINE

This user's manual describes the specifications and notes when using the option -lang=cpp14, which allows the compiler to compile the source program with C++14 standard and is a part of the C compiler package for RL78 family CC-RL V1.14.00.

Please also refer to the CC-RL User's Manual as well.

1.1 Feedback on the option -lang=cpp14

Please send your feedback on this feature from the URL below:

https://forms.office.com/r/wSGqp6BKic

1.2 Copyrights

This software uses the following softwares.

- LLVM and Clang are copyrights of University of Illinois at Urbana-Champaign.
- Protocol Buffers is copyright of Google Inc.

The libraries for C++ use the following softwares. Please refer to the license files included in the compiler package for detail.

- compiler_rt
- libc++
- libc++abi
- newlib

Other software components are copyright of Renesas Electronics Corporation.

2. OPTIONS

Specify the following option for compiling a source program with C++14 standard.

```
-lang=cpp14
```

[Detailed description]

This option allows the compiler to compile a source program with C++14 standard (ISO/IEC 14882:2014).

- A compile error will occur when C source files are specified as input with this option. For details of the kind of input/output files, please refer to "2.2 I/O Files" in CC-RL User's Manual.
- Please refer to the following section for the existing options that can be used with this option.

2.1 Existing options available under the C++14 specification

2.1.1 Compile options

This section shows the existing compile options allowed to combine with -lang=cpp14. "X" in the "Combinable" column indicates that the option is just ignored, or an error message will be output.

Table 1 Compiler options available under the C++14 standard

Category	Option	Combinable	Note
Version display specification	-V	✓	
Help display specification	-help	✓	
Output file specification	-o	✓	
	-obj_path	✓	
	-asm_path	✓	
	-prep_path	✓	
Source debugging control	-g	✓	Some debug information of C++ standard specifications will be discarded. Please refer to the section "NOTES" below.
	-g_line	✓	
Device specification relation	-cpu	✓	
	-use_mda	✓	
Processing interrupt specification	-P	✓	
	-S	✓	
	-c	✓	

Table 1 Compiler options available under the C++14 standard (2)

Category	Option	Combinable	Note
Preprocessor control	-D	✓	
	-U	✓	
	-I	✓	
	-preinclude	✓	
	-preprocess	X	
Memory model	-memory_model={small | medium}	✓	
	-far_rom	✓	Internal error may occur in some programs.
Optimization	-O{ size | speed | default | lite | nothing }	✓	
	-goptimize	✓	
Optimization (detailed)	-Oinline_level[=*value*]	✓	
	-Oinline_size[=*value*]	✓	
	-Opipeline[={on|off}]	✓	
	-Ounroll[=*value*]	✓	
	-Otail_call[={on|off}]	✓	
	-Odelete_static_func[={on|off}]	✓	
	-Omerge_files	X	
	-Ointermodule	X	
	-Owhile_program	X	
	-Oalias={ansi|noansi}	X	
	-Osame_code={on|off}	✓	
Additional information output	-cref	X	
	-pass_source	✓	
Error output control	-no_warning_num	✓	Applicable to the messages ranged for W0510000-W0519999 and W0530000-W0559999 (W0520000-W0529999 are not output when -lang=cpp14 is specified).
	-change_message	✓	Applicable to the messages ranged for W0510000-W0519999 and W0530000-W0549999 (W0520000-W0529999 are not output when -lang=cpp14 is specified)
	-error_file	✓	

Table 1 Compiler options available under the C++14 standard (3)

Category	Option	Combinable	Note
Code generation changing	-dbl_size={4|8}	✓	
	-signed_char	✓	
	-signed_bitfield	X	This option has no effect when -lang=cpp14 is specified: The bitfield type for which neither "signed" nor "unsigned" is specified as "signed". This is different from the interpretation when -lang=c or -lang=c99 is specified: those options handle the bitfield type for which neither "signed" nor "unsigned" is specified as "unsigned".
	-switch	✓	
	-volatile	X	
	-merge_string	X	
	-pack	✓	
	-stuff	X	
	-stack_protector -stack_protector_all	X	
	-insert_nop_with_label	X	
	-control_flow_integrity	X	
Extensions	-strict_std	X	
	-refs_without_declaration	X	
	-large_variable	X	
	-nest_comment	X	
	-character_set	X	This option has no effect when -lang=cpp14 is specified: The encoding in the source file is always interpreted as UTF-8.
MISRA check	-misra2004	X	
	-misra2012	X	
	-ignore_files_misra	X	
	-check_language_extension	X	
	-misra_intermodule	X	
Subcommand file specification	-subcommand	✓	
Assembler and linker control	-asmopt=*arg*	✓	
	-lnkopt=*arg*	✓	
	-asmcmd=*filename*	✓	
	-lnkcmd=*filename*	✓	
	-dev=*filename*	✓	
Compiler transition support	-convert_cc={ca78k0r|nc30|iar}	X	
	-unaligned_pointer_for_ca78k0r	X	

2.1.2 Assemble options

All existing assemble options are allowed to combine with -lang=cpp14.

2.1.3 Link options

The existing link option below is not allowed to combine with -lang=cpp14. The other options are allowed to combine with -lang=cpp14.

Table 2 Link options available under the C++14 standard

Category	Option	Combinable	Note
Output control	-VFINFO	X	

3. COMPILER LANGUAGE SPECIFICATIONS

3.1 Basic language specifications

3.1.1 Unsupported C++ language specifications

The following language specifications are not supported.

- Exception handling
- Runtime type identification
- Threads
- Atomic operations

3.1.2 Implementation-defined behavior of C++14

This section covers the implementation-defined behavior.

Table 3 Implementation-defined behavior

Section No.	Item	Description
1.3.6	diagnostic message	Refer to "7. Message".
1.4	required libraries for freestanding implementation	Refer to "5. Library Specifications".
1.7	bits in a byte	8 bits.
1.9	interactive device	What constitutes an interactive device is not specified.
1.10	number of threads in a program under a freestanding implementation	Multi-threaded execution is not supported.
2.2	mapping physical source file characters to basic source character set	Map as UTF-8 as-is.
2.2	physical source file characters	UTF-8.
2.2	converting characters from source character set to execution character set	The source character set and the execution character set are the same.
2.2	whether source of translation units must be available to locate template definitions	The source is not required.
2.9	mapping header name to header or external source file	Interpreted as described and mapped to a file name.
2.14.3	value of multicharacter literal	the lower 4 bytes of the execution character set.
2.14.3	value of wide-character literal containing multiple characters	the last character in the execution character set.
2.14.3	value of wide-character literal with single c-char that is not in execution wide-character set	the value of the character.
2.14.3	encoding of universal character name not in execution character set	the value of the character.
2.14.3	semantics of non-standard escape sequences	¥e and ¥E are valid. Both values are 0x1b.
2.14.3	value of character literal outside range of corresponding type	An error occurs.
2.14.5	concatenation of some types of string literals	An error occurs.

Table 3 Implementation-defined behavior

Section No.	Item	Description
3.6.1	defining main in freestanding environment	Not defined.
3.6.1	parameters to main	Not defined.
3.6.1	start-up and termination in freestanding environment	Not defined. Depends on the startup routine.
3.6.1	linkage of main	C linkage.
3.6.2	dynamic initialization of static objects before main	Depends on the startup routine.
3.6.2	dynamic initialization of thread-local objects before entry	Threads are not supported.
3.9.1	extended signed integer types	Extended signed integer types are not supported.
3.9.1	representation of char	1 byte.
3.9.1	signedness of char	Unsigned char type. However, it can be switched to signed char type by -signed_char option.
3.9.1	value representation of floating-point types	Compliant with IEEE754.
3.9.2	value representation of pointer types	Refer to "3.1.3 Internal representation and value area of data".
3.11	alignment	Refer to "3.1.3 Internal representation and value area of data".
4.13	rank of extended signed integer type	Extended signed integer type is not supported.
5.3.3	sizeof applied to fundamental types other than char, signed char, and unsigned char	Refer to "3.1.3 Internal representation and value area of data".
5.3.4	support for over-aligned types	Over-aligned types are not supported.
5.8	result of right shift of negative value	Arithmetic shift is performed.
7.2	underlying type for enumeration	Refer to "3.1.3 Internal representation and value area of data".
7.4	meaning of asm declaration	The asm declaration is not supported.
8.4.1	string resulting from __func__	A function name is returned.
16.2	nesting limit for #include directives	The nesting limit depends on the memory available.
16.6	#pragma	Refer to "Pragma directive".
16.8	text of __DATE__ when date of translation is not available	The date is always available.
16.8	text of __TIME__ when time of translation is not available	The time is always available.
16.8	definition and meaning of __STDC__	Defined as 1.
16.8	definition and meaning of __STDC_VERSION__	Not defined.
17.6.5.12	exceptions thrown by standard library functions that do not have an exception specification	Exceptions are not supported.
18.2	type of size_t	unsigned int.
18.5	exit status	Not defined.

3.1.3 Internal representation and allocation of data

This section describes the internal representation and value range for each data type in CC-RL.

(1) Basic type

Table 4 Basic types

Data Type	Size (byte)	Alignment (byte)	Signed/ Unsigned	Data range		Note
				Minimum Value	Maximum Value	
char	1	1	Unsigned	0	+255	The value range is the same as that of signed char when -signed_char is specified.
signed char	1	1	Signed	-128	+127	
unsigned char	1	1	Unsigned	0	+255	
short	2	2	Signed	-32768	+32767	
signed short	2	2	Signed	-32768	+32767	
unsigned short	2	2	Unsigned	0	+65535	
int	2	2	Signed	-32768	+32767	
signed int	2	2	Signed	-32768	+32767	
unsigned int	2	2	Unsigned	0	+65535	
long	4	2	Signed	-2147483648	+2147483647	
singed long	4	2	Signed	-2147483648	+2147483647	
unsigned long	4	2	Unsigned	0	+4294967295	
long long	8	2	Signed	-9223372036 854775808	+9223372036 854775807	
signed long long	8	2	Signed	-9223372036 854775808	+92233720368 54775807	
unsigned long long	8	2	Unsigned	0	+1844674407 3709551615	

Table 4 Basic types (2)

Data Type	Size (byte)	Alignment (byte)	Signed/ Unsigned	Data range		Note
				Minimum Value	Maximum Value	
wchar_t	2	2	Unsigned	0	+65535	
char16_t	2	2	Unsigned	0	+65535	
char32_t	4	2	Unsigned	0	+4294967295	
bool	1	1	Unsigned	-	-	Only the bit 0 is meaningful. The bits from 1 to 7 are undefined.
float	4	2	Signed	1.17549435E-38F	3.40282347E+38F	
double (-double_size=4)	4	2	Signed	1.17549435E-38F	3.40282347E+38F	
double (-double_size=8)	8	2	Signed	2.2250738585072 014E-308	1.7976931348623 158E+308	
long double (-double_size=4)	4	2	Signed	1.17549435E-38F	3.40282347E+38F	
long double (-double_size=8)	8	2	Signed	2.2250738585072 014E-308	1.79769313486 23158E+308	

(2) Derived types

- Pointer and array types

Table 5 Pointer and array types

Data Type		Size(byte)	Alignment(byte)
Pointer type	near pointer	2	2
	far pointer	4	2
Lvalue reference type	near reference	2	2
Rvalue reference type	far reference	4	2
Pointer to data member type		2	2
Pointer to member function type		4	2
Array type		The size of the element type * The number of the elements	The alignment of the element type

- Enumeration type

Table 6 Enumeration type

The minimum value for enumerator	The maximum value for enumerator	Underlying type	Note
-128	127	signed char	-
0	255	unsigned char	If all enumerators are in the range 0-255, this representation applies.
-32768	32767	signed short	-
0	65535	unsigned short	If all enumerators are in the range 0-65535, this representation applies.
-2147483647	2147483647	signed long	-
0	4294967295	unsigned long	If all enumerators are in the range 0-4294967295, this representation applies.
-9223372036854775808	9223372036854775807	signed long long	-
0	18446744073709551615	unsigned long long	If all enumerators are in the range 0-18446744073709551615, this representation applies.
Otherwise		signed long long	A warning will be output.

3.2 Language extension specifications

3.2.1 Reserved words

Please refer to "4.2.1 Reserved words" in the CC-RL User's Manual for detail of the keywords reserved by CC-RL.

However, the following reserved words are not supported.

- __saddr
- __callt
- __sectop
- __secend

Some of the specifications for the following reserved word differ from those when the -lang=c99 option is specified.

- __inline
  When the -lang=cpp14 option is specified, the keyword __inline is an alias for the keyword inline; this is for compliance with the specification of inline for C++.

3.2.2 Macros

The Table 6 shows the macros whose definitions differ along with the parameter given for the option -lang. Please also refer to "4.2.2 Macros" in the CC-RL User's Manual as well for detail of the other macros.

Note that the values in the table are in decimal.

Table 7 Macros

Name	Definition when -lang=cpp14 is specified	Definition when -lang=c or -lang=c99 is specified
__cplusplus	201402L	Undefined
__clang__	1	Undefined
__STDC_HOSTED__	0	0 (when -lang=c99 is specified)
__STDC__	1	1 (when -strict_std is specified)
__STDC_VERSION__	Undefined	199409L(when both -lang=c and -strict_std are specified) 199901L (when -lang=c99 is specified)
__STDC_IEC_559__	1	1 (when -lang=c99 is specified)

3.2.3 #pragma directives

#pragma directives described in "4.2.4 #pragma directives" in the CC-RL User's Manual are not supported.

3.2.4 Intrinsic functions

Intrinsic functions described in "4.2.7 Intrinsic functions" in the CC-RL User's Manual are supported. Please refer to the CC-RL User's Manual for details.

4. SECTION SPECIFICATIONS

This section describes the names and the relocation attributes of the reserved sections when compiling under C++14 language specifications. Please refer to the CC-RL User's Manual for the other sections.

4.1 Section name

Table 8 Reserved section names

Default Section Name	Relocation Attribute	Description
.init_array	CONSTF	Section for the global constructors
.callt0	CALLT0	Section for the table used when callt functions called
.text	TEXT	Section for code (allocated to the near area)
.textf	TEXTF	Section for code (allocated to the far area)
.textf_unit64kp	TEXTF_UNIT64KP	Section for code (section is allocated so that the start address is an even address and the section does not exceed the (64 Kbytes - 1) boundary)
.const	CONST	ROM data (allocated to the near area) (within the mirror area)
.constf	CONSTF	ROM data (allocated to the far area)
.data	DATA	Section for near initialized data (with initial value)
.dataf	DATAF	Section for far initialized data (with initial value)
.sdata	SDATA	Section for initialized data (with initial value, allocated to saddr)
.bss	BSS	Section for data area (without initial value, allocated to near area)
.bssf	BSSF	Section for data area (without initial value, allocated to far area)
.sbss	SBSS	Section for data area (without initial value, allocated to saddr)
.option_byte	OPT_BYTE	Section specific for user option byte and on-chip debugging specification
.security_id	SECUR_ID	Section specific for security ID specification
.flash_security_id	FLASH_SECUR_ID	Section specific for flash programmer security ID specification
.vect<vector table address>	AT	Interrupt vector table If the -split_vect option is specified, a section is generated based on ".vect<vector table address>". The vector table address is in hexadecimal notation

5. LIBRARY SPECITICATIONS

5.1 Outline

The CC-RL provides the dedicated libraries for compiling C++ source programs based on the software below. Please refer to the source program included in the compiler package for detail.

- compiler_rt
- libc++
- libc++abi
- newlib

5.2 Supplied Libraries

The following 6 libraries are provided for each of the CPU core types S1, S2, and S3 specified by the option -cpu. All these libraries are dedicated for uses with -lang=cpp14 specified, and does not supported uses with -lang=c or -lang=c99.

Table 9 Supplied libraries

Library Name	Outline
rl78_libc.lib	The standard library (C99)
rl78_libm.lib	The standard math library (C99)
rl78_libgloss.lib	The low-level library
rl78_libcxx.lib	The standard library (C++14)
rl78_libcxxabi.lib	The runtime library for ABI support (C++14)
rl78_compiler-rt.lib	The runtime library for the compiler

* The libraries corresponding for the S2 core are built with multiplier and divider/multiply-accumulator enabled (-use_mda=mda).

* All the libraries assume single precision for the "double" and "long double" floating point type (-dbl_size=4).

5.3 Header Files

The header files required for using the C++ libraries are listed below.

Table 10 Header Files

Category	File Name	Description
The standard library (C99)	<assert.h>	Header file for program diagnostics
	<complex.h>	Header file for complex number
	<ctype.h>	Header file for character conversion and classification
	<errno.h>	Header file for reporting error condition

Table 10 Header Files (2)

Category	File Name	Description
The standard library (C99)	<float.h>	Header file for floating-point representation and operation
	<inttypes.h>	Header file for the maximum-width integer type
	<iso646.h>	Header file for alternative spellings of macro names
	<limits.h>	Header file for quantitative limiting of integers
	<locale.h>	Header file for localization
	<math.h>	Header file for mathematical calculation
	<setjmp.h>	Header file for non-local jump
	<signal.h>	Header file for signal handling
	<stdarg.h>	Header file for supporting functions having variable arguments
	<stdbool.h>	Header file for logical types and values
	<stddef.h>	Header file for common definitions
	<stdint.h>	Header file for integer type of the specified width
	<stdio.h>	Header file for standard I/O
	<stdlib.h>	Header file for general utilities
	<string.h>	Header file for manipulation of sequential memory and character string
	<tgmath.h>	Header file for type generic mathematical calculation
	<wchar.h>	Header file for utilities related to multibyte/wide character
	<wctype.h>	Header file for wide character conversion and classification
The standard library (C++14)	<algorithm>	Header file for algorithmic operations
	<array>	Header file for fixed sized sequential container
	<bitset>	Header file for fixed sized sequential bit container
	<chrono>	Header file for date and time
	<codecvt>	Header file for character code conversion
	<complex>	Header file for complex number
	<condition_variable>	Header file for synchronization among the threads

Table 10 Header Files (3)

Category	File Name	Description
The standard library (C++14)	<deque>	Header file for double ended queue
	<forward_list>	Header file for singly-linked list
	<fstream>	Header file for file stream
	<functional>	Header file for function object
	<future>	Header file for providing "future" pattern
	<initializer_list>	Header file for initializer list
	<iomanip>	Header file for I/O manipulator and formatting
	<ios>	Header file for base classes of iostream
	<iosfwd>	Header file for forward declaration of iostream
	<iostream>	Header file for standard iostream objects
	<istream>	Header file for input streams
	<iterator>	Header file for iterators
	<limits>	Header file for properties of the implementation's representation of the arithmetic type
	<list>	Header file for doubly-linked list
	<locale>	Header file for the information peculiar to a locale
	<map>	Header file for associative container of unique keys and values
	<memory>	Header file for memory management
	<mutex>	Header file for mechanisms for mutual exclusion
	<new>	Header file for dynamic storage allocation
	<numeric>	Header file for generalized numeric operations
	<ostream>	Header file for output streams
	<queue>	Header file for queue
	<random>	Header file for random number generation
	<ratio>	Header file for compile time rational arithmetic
	<regex>	Header file for regular expression template
	<scoped_allocator>	Header file for scoped allocator
	<set>	Header file for associative container of unique keys

Table 10 Header Files (4)

Category	File Name	Description
The standard library (C++14)	<sstream>	Header file for string stream
	<stack>	Header file for stack
	<streambuf>	Header file for stream buffers
	<string>	Header file for string classes
	<system_error>	Header file for system error support
	<tuple>	Header file for tuples
	<type_traits>	Header file for type traits
	<typeindex>	Header file for type indexes
	<unordered_map>	Header file for unordered associative containers of unique kyes and values
	<unordered_set>	Header file for unordered associative containers of unique keys
	<utility>	Header file for utility components
	<valarray>	Header file for numeric arrays
	<vector>	Header file for vector
The C compatible standard libraries	<cassert>	Header file compatible with assert.h
	<ccomplex>	Header file compatible with complex.h
	<cctype>	Header file compatible with ctype.h
	<cerrno>	Header file compatible with errno.h
	<cfloat>	Header file compatible with float.h
	<cinttypes>	Header file compatible with inttypes.h
	<ciso646>	Header file compatible with iso646.h
	<climits>	Header file compatible with limits.h
	<clocale>	Header file compatible with locale.h
	<cmath>	Header file compatible with math.h
	<csetjmp>	Header file compatible with setjmp.h
	<csignal>	Header file compatible with signal.h
	<cstdarg>	Header file compatible with stdarg.h
	<cstdbool>	Header file compatible with stdbool.h
	<cstddef>	Header file compatible with stddef.h
	<cstdint>	Header file compatible with stdint.h
	<cstdio>	Header file compatible with stdio.h
	<cstdlib>	Header file compatible with stdlib.h
	<cstring>	Header file compatible with string.h
	<ctgmath>	Header file compatible with tgmath.h
	<cwchar>	Header file compatible with wcchar.h
	<cwctype>	Header file compatible with wctype.h

6. STARTUP

6.1 Startup Routine

Before entering the main function, execute the following processes in addition to those described in 8.2 Startup Routine in CC-RL User's Manual.

6.1.1 Initialization of global objects of class type

Call the constructor for each object of class type declared with static storage duration.

The addresses of those constructors are stored in the section named .init_array. Put the following description in the startup routine for calling all of them.

```
        MOVW    BC,#LOWW(SIZEOF(.init_array))
        BR      $.L2_INIT
    .L1_INIT:
        DECW    BC
        DECW    BC
        MOV     ES,#HIGHW(STARTOF(.init_array))
        MOVW    AX,ES:LOWW(STARTOF(.init_array))[BC]
        MOV     CS,#0x00
        PUSH    BC
        CALL    AX
        POP     BC
    .L2_INIT:
        CLRW    AX
        CMPW    AX,BC
        BNZ     $.L1_INIT
```

7. MESSAGE

7.1 Message Formats

There are two formats of message when -lang=cpp14 is specified.

7.1.1 Format 1

This kind of format contains a message number as explained in "10 MESSAGE" in CC-RL User's Manual. Please refer to the manual for detail.

Those messages are numbered as:

0510000-0519999, 0530000-0539999, 0540000-0549999, 0550000-0559999, and 0560000-0569999.

(1) When the file name and line number are included

```
file-name (line-number) : message-type 05 message-number : message
```

(2) When the file name and line number aren't included

```
message-type 05 message-number : message
```

7.1.2 Format 2

This kind of format is output as below.

(1) When the file name, line number, and column number are included

```
file-name : line-number : column-number : message-type : message
```

(2) When neither the file name, line number, nor column number are included

```
message-type : message
```

7.2 Message Types

The message types are as follows.

Table 11 Message Types

Message Type		Description
Format 1	Format 2	
C	-	Internal error : Processing is aborted. No object codes are generated.
E	error	Error : Processing is aborted if a set number of errors occur. No object codes are generated.
F	fatal	Fatal error : Processing is aborted. No object codes are generated.
M	remark	Information : Processing continues. Object codes are generated.
W	warning	Warning : Processing continues. Object codes are generated (They might not be what the user intended).
-	note	Additional information for the other types of messages.

8. NOTES

8.1 Missing information for source level debugging

Information for source level debugging for the language specification listed below is not supported.

- Anonymous unions
- Namespaces
- Derived classes
  - Virtual base classes
  - Virtual functions
- Templates

8.2 sbrk

The standard library calls the function sbrk in processing such as dynamic memory management. The function is included in low level support library rl78_libgloss.lib and defined as below. If you prefer sbrk to work differently, please implement your own sbrk and link it to your application.

```
  #define HEAPSIZE 0x400
  union HEAP_TYPE {
      signed long dummy;
      signed char heap[HEAPSIZE];
  };

  static union HEAP_TYPE heap_area ;
  static signed char *brk=(void *)&heap_area;
  void *sbrk(int size)
  {
      void *p = 0;
      if (brk + size > heap_area.heap + HEAPSIZE) {
          return (void *)-1;
      }
      p = brk;
      brk += size;
      return p;
}
```

The copyright of the program above is reserved by Renesas Electronics Corporation. Please note the dislaimer below as well.

Revision History	CC-RL C++ User’s Manual

Rev.	Date	Description	
		Page	Summary
1.00	Jan.20.23	—	First Edition issued
1.01	Jul.20.23	4	Feedback on the Technical Preview Edition is added.
		13	Table 6 is changed.
		13	Unsupported reserved words are added.
		14	Value of __STDC_IEC_559__ is changed.
1.02	Apr.20.24	—	Removed the descriptions “Technical Preview Edition” from the entire document as we update it for CC-RL V1.14.00.
		16	Unsupported header files(time.h, ctime, exception, stdexcept) are deleted.
		22	“8.2 sbrk” is added.

CC-RL
C++ User’s Manual

Publication Date: Rev.1.02 Apr.20.24

Published by: Renesas Electronics Corporation

CC-RL
C++
User's Manual

Renesas Electronics Corporation

R20UT5266EJ0102